DMaC: Distributed Monitoring and Checking

Wenchao Zhou, Oleg Sokolsky, Boon Thau Loo, Insup Lee

University of Pennsylvania

This work was partially supported by ONR MURI N00014-07-0907, NSF CNS-0721845 and NSF IIS-0812270.

Introduction

- **Proliferation of new network architectures and protocols**
  - Overlay networks with new capabilities
    - Mobility, resiliency, anycast, multicast, anonymity, etc
  - Distributed data management applications
    - Network monitoring, publish-subscribe systems, content-distribution networks
- **Declarative networking**
  - Declarative query language for network protocols [SIGCOMM 05, SIGMOD 06]
  - Compiled to distributed dataflows, executed by distributed query engine
  - Performance comparable to imperative implementations
  - *Orders of magnitude reduction* in code size

Requirements for Verification

- **Requirement for verification of user-defined properties**
  - Behavioral: sequencing of events, correlation between values
  - Timing / Performance: route oscillation, slow convergence
  - Enforce trust management / access control policies
- **However, existing approaches are often…**
  - Platform dependent, hard to be generalized
  - Specified at the implementation level, formal reasoning are not possible
- **Growing interest in formal tools and programming frameworks**

Formal Methods

- **Formal verification techniques**
  - Model Checking: formal, exhaustive, but doesn't scale well
  - Testing: informal, non-exhaustive, no guarantee for given executions
- **Runtime verification**
  - Light-weight verification technique
  - Check a current program execution against its formal properties at runtime
  - Advantages:
    - Property checking on a trace is easier than over an arbitrary model
    - Validate implementation directly - guarantee for current execution

Contributions

- **A framework of distributed runtime verification & its deployment**
  - Independent of monitored distributed systems
  - Seamlessly integrated within a declarative networking engine
  - Generate runtime checkers and deploy them across the network
- **Translation from formal specifications to declarative networks**
  - Automatic compilation of formal specifications to distributed queries
  - Opportunity of applying existing database query optimizations for efficient plan generation and dynamic re-optimization
- **Implementation and experimental evaluation on a local cluster**
  - Feasibility of the approach, in terms of performance overhead
  - Functionality of the property specification language

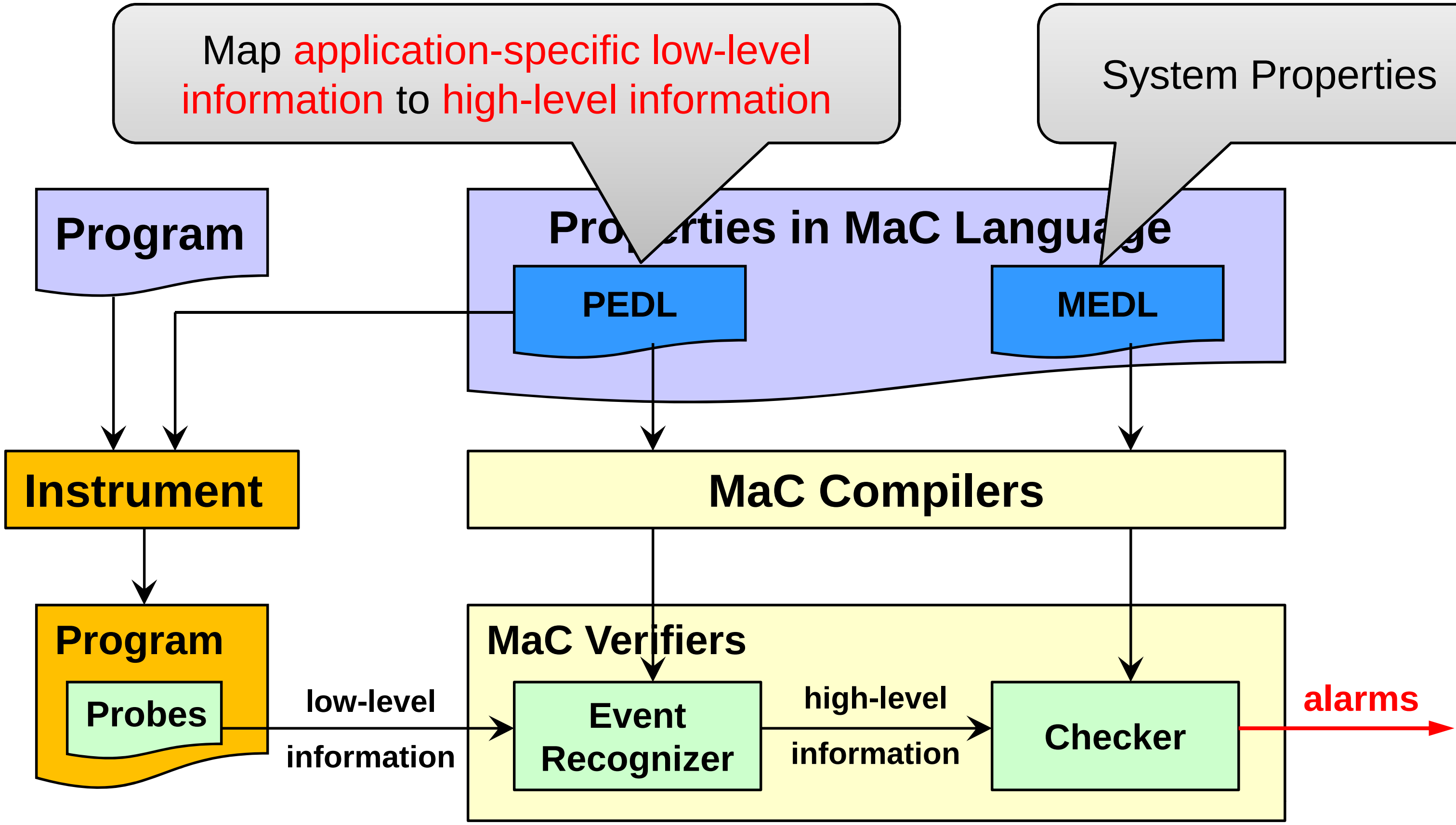
Outline of Talk

- Introduction
- Background and Motivation
  - Runtime Monitoring and Checking
  - Declarative Networking
- Architectural Overview of DMaC
- Compilation to Declarative Networking Queries
- Experimental Evaluation
- Conclusion & Future Work

MaC: Monitoring and Checking

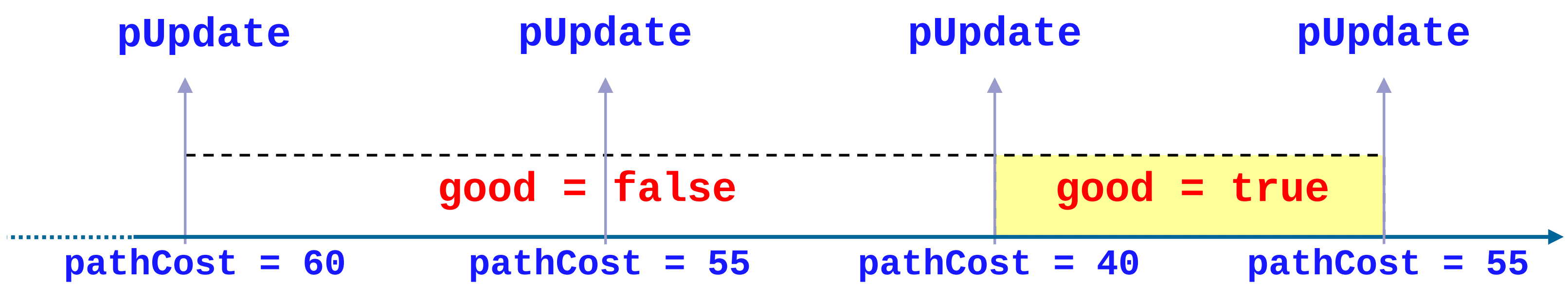
- **A runtime verification framework**
  - Languages for monitoring and checking properties
  - Architecture for run-time verification
  - Prototype implementation: Java-MaC, etc.
- **PEDL (Primitive Event Definition Language)**
  - Low-level specification, Dependent on underlying applications
  - Event recognition based on the events gathered from monitored systems
  - Interface between monitored systems and MEDL
- **MEDL (Meta Event Definition Language)**
  - Independent of the monitored system
  - Express requirements using events and conditions
  - Describe the safety requirements

MaC Architecture



MEDL Specification Language

- **Monitoring properties: instantaneous vs. durational**
- **Events**
  - Instantaneous incidents
  - such as variable updates event pUpdate = update(pathCost)
- **Conditions**
  - Proposition about the program
  - May be ***true*** **/** ***false*** **/** ***undefined*** for a duration of time
  - *such as condition good = pathCost < 50*



MEDL Specification Language (cont.)

- **Auxiliary variables**
  - Updated in response to events
  - For more complex events, e.g. count the occurrences of a specific event

- **Composition of events and conditions**

$$E ::= e \mid \text{start}(C) \mid \text{end}(C) \mid E_1 \vee E_2 \mid E_1 \wedge E_2 \mid E \text{ when } C$$

$$C ::= c \mid [E_1, E_2) \mid \neg C \mid C_1 \vee C_2 \mid C_1 \wedge C_2$$

- **Capable of expressing complex user-defined properties**

Limitations of Centralized Monitoring

- **Centralized monitoring and checking**
  - Observed events are sent to a global monitor
  - Cross-node communication to feed base events
  - Some properties are intrinsically distributed, e.g. network properties within administrative domains

Can we implement Distributed MaC? Ideally, the distributed deployment can be leveraged using existing infrastructures.

Declarative Networking

- **Declarative query language for network protocols**
  - Easy distribution and cross-node communication
  - Network Datalog (NDlog) – distributed Datalog
  - *Location specifiers* (@ symbol) indicate the source/destination of messages

- **Example: Network Reachability**

r1: reachable(@S,D) :- link(@S,D)

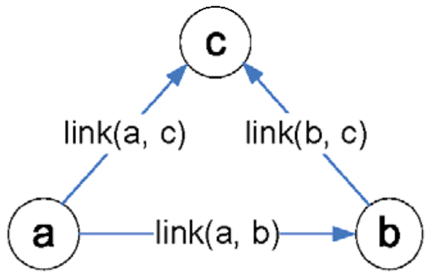
r2: reachable(@S,D) :- link(@S,Z), reachable(@Z,D)

link(@a,b) – "there is a link from node *a* to node *b*"

reachable(@a,b) – "node *a* can reach node *b*"

If there is a link from S to D, then S can reach D.

If there is a link from S to Z, AND Z can reach D, then S can reach D.



Node a	Node b
link(@a, b)	link(@b, c)
link(@a, c)	reachable(@b, c)
reachable(@a, c)	

Natural Match - MEDL and NDlog

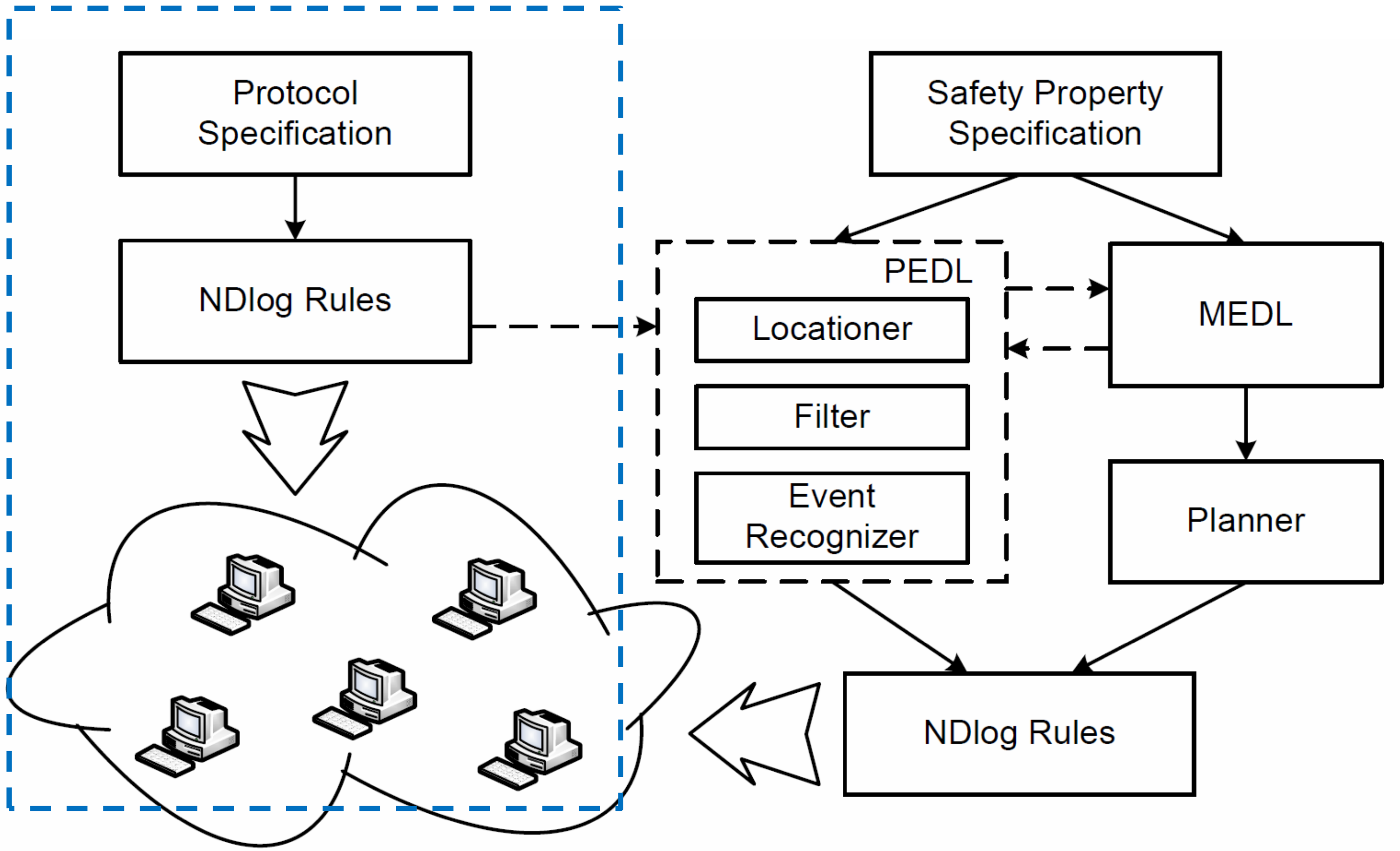
- **Similar notion as event, condition, and auxiliary variable**
  - Tuples without materialization
  - Explicitly stated materialized tables
- **Support for composition of events and conditions**

```
materialize(reachable, infinity, keys(1,2)).
materialize(link, infinity, keys(1,2)).
r0: link(@S,D) :- discovery(@S,D).
r1: reachable(@S,D) :- link(@S,D).
r2: reachable(@S,D) :- link(@S,Z), reachable(@Z,D).
```

Outline of Talk

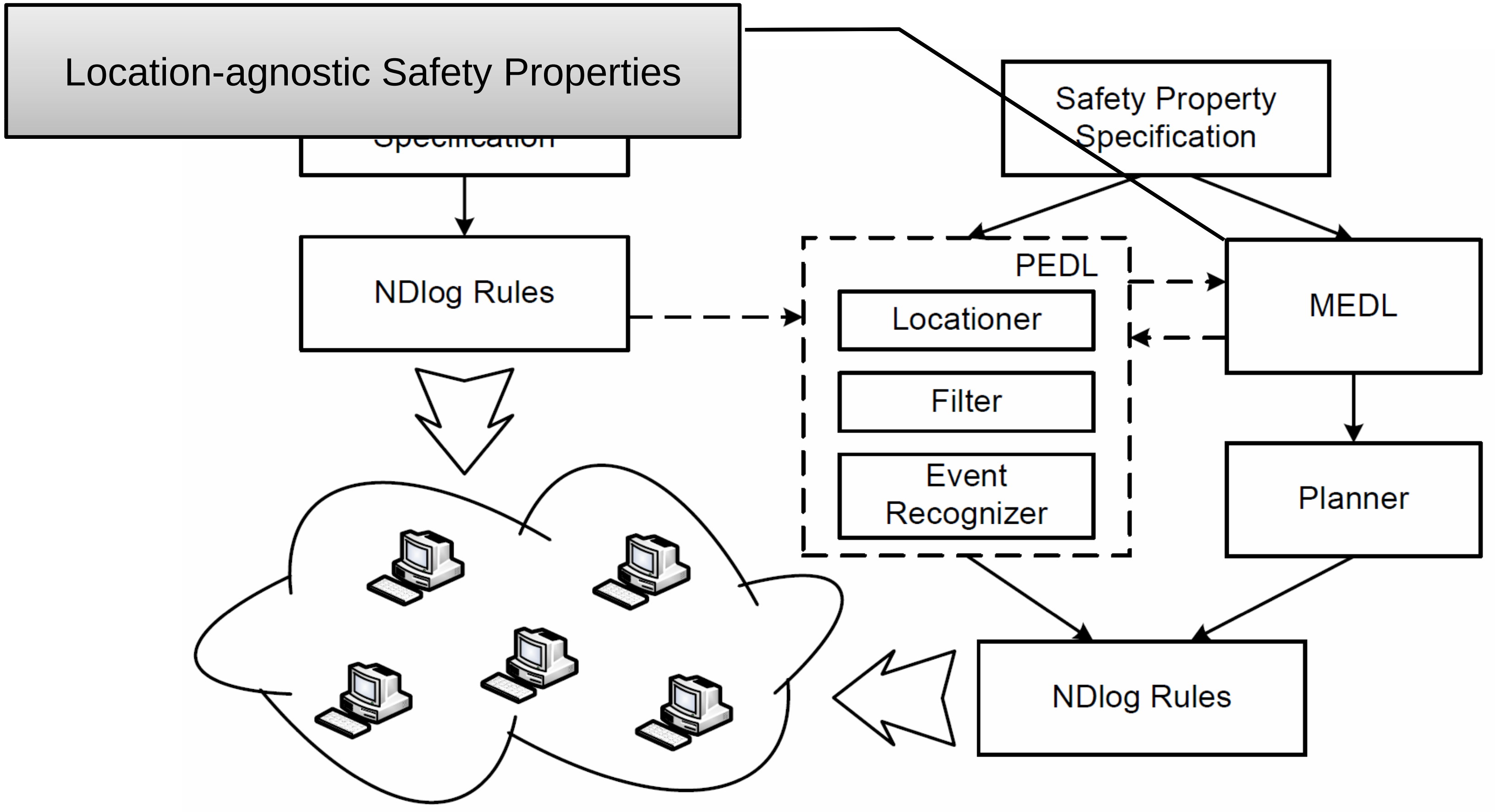
- Introduction
- Background and Motivation
- DMaC Architectural Overview
- Compilation to Declarative Networking Queries
- Experimental Evaluation
- Conclusion & Future Work

DMaC Architecture

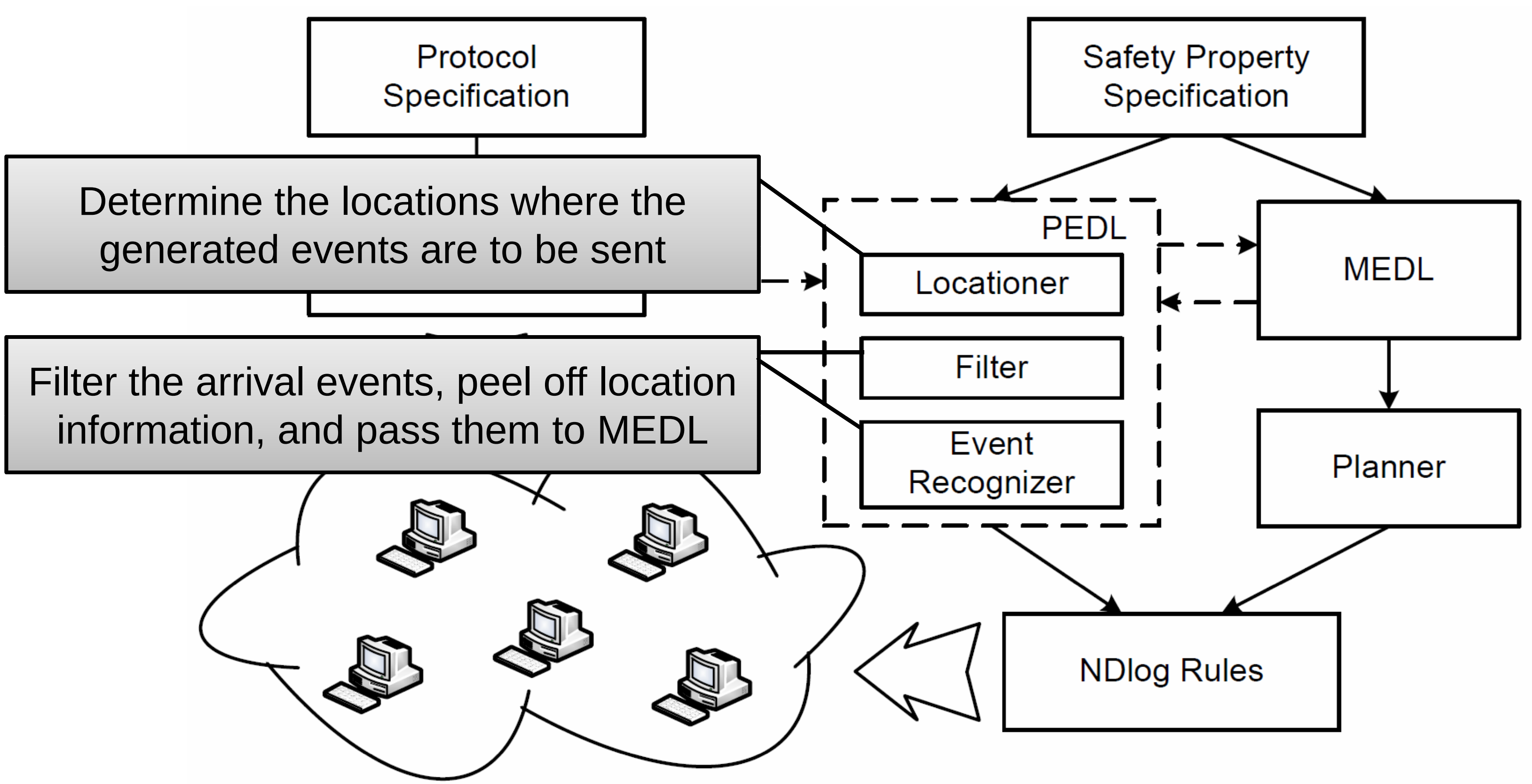


Declarative Networks / Legacy Networks

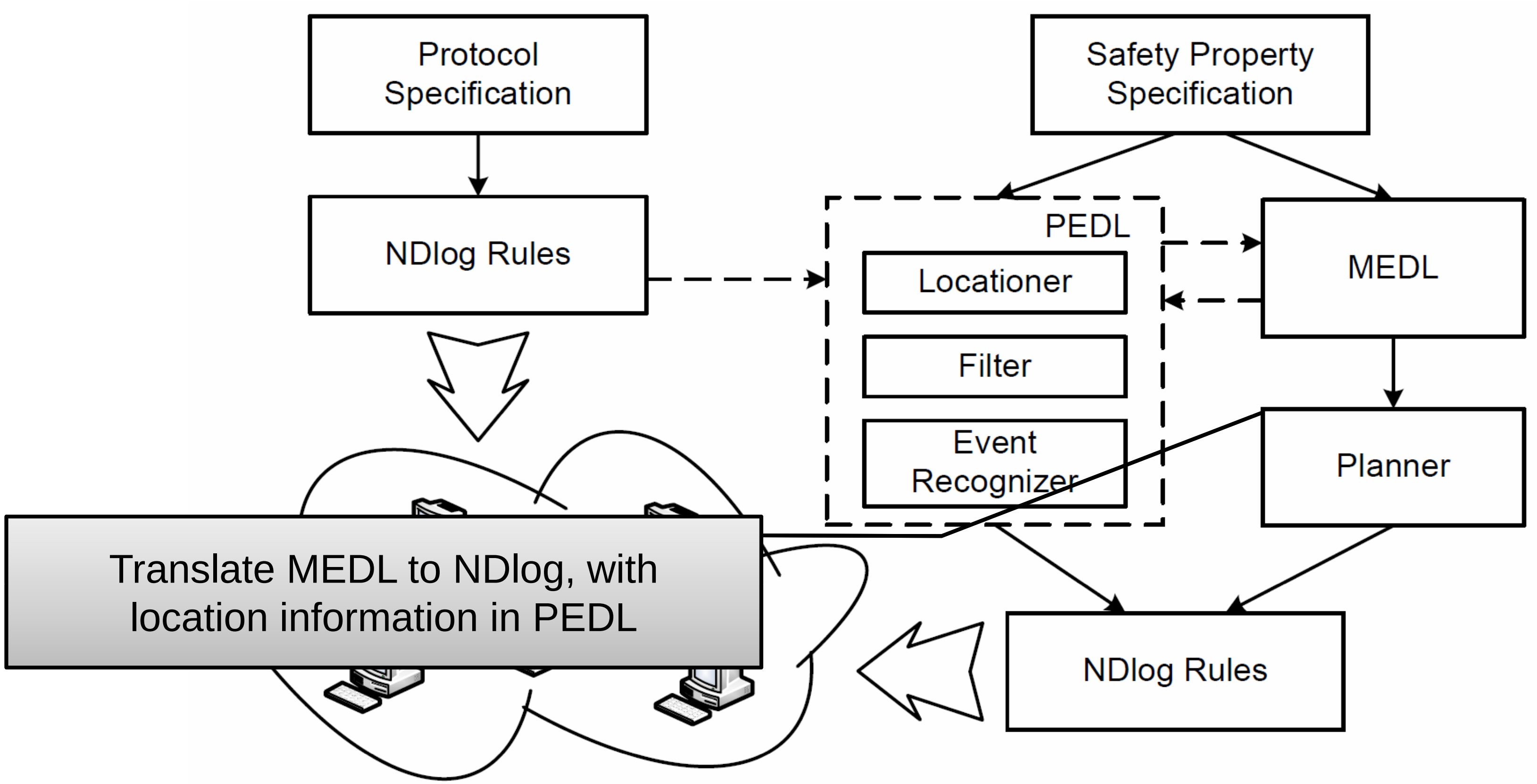
DMaC Architecture



DMaC Architecture

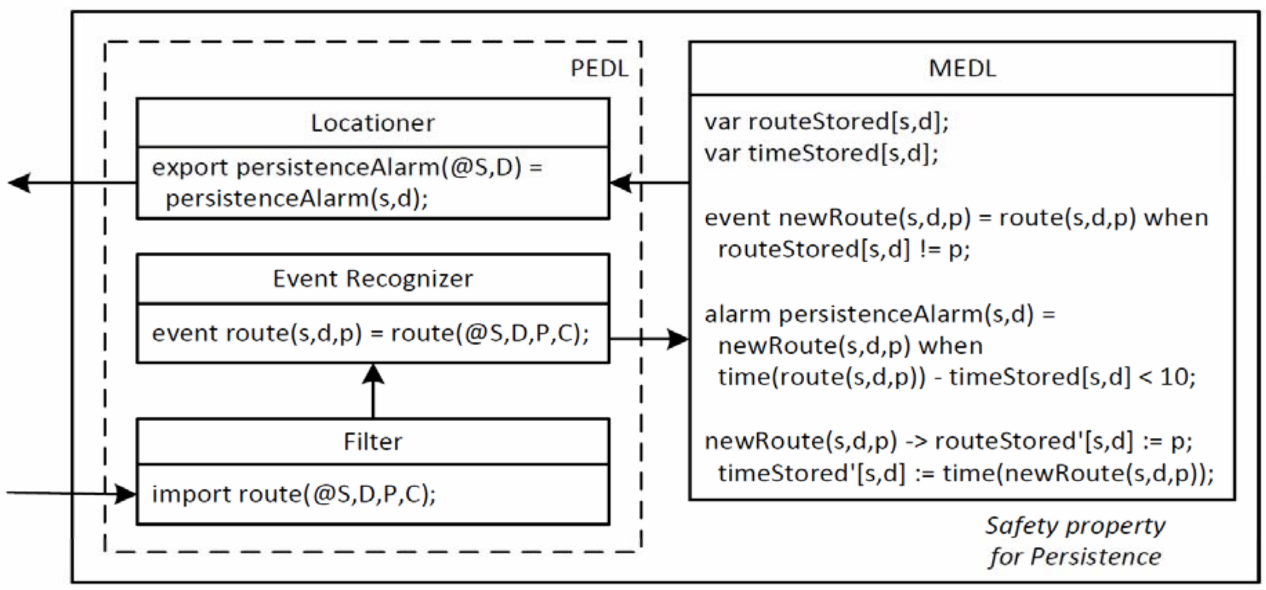


DMaC Architecture



Example: Route Persistence Property

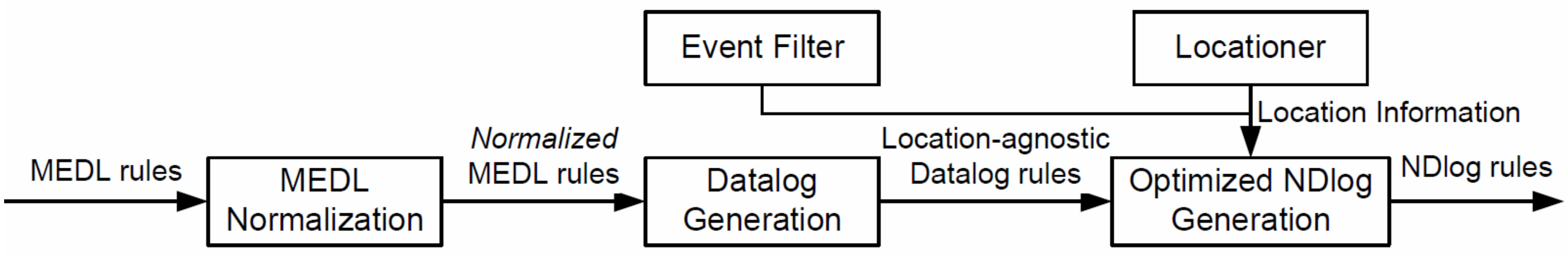
- **Route persistence property**
  - Track the duration that a computed route persists without changing
  - Raise *persistenceAlarm* when changes occur too quickly
    - Could be a symptom of more serious issues, e.g. lack of convergence



Outline of Talk

- Introduction
- Background and Motivation
- DMaC Architectural Overview
- Compilation to Declarative Networking Queries
  - Translation from MEDL to Datalog
  - Optimization of Generated NDlog Program
- Experimental Evaluation
- Conclusion & Future Work

Compile MEDL into NDlog



- **MEDL normalization**
  - Rewrite MEDL rules into normalized MEDL expressions
  - Event / condition = an application of exactly one operator
- **Datalog generation**
  - Rewrite normalized MEDL expressions into location-agnostic Datalog rules
- **Optimized NDlog generation**
  - Tag location information

Datalog Generation

Normalized MEDL rules are rewritten into location-agnostic Datalog rules

MEDL Rules	Corresponding Datalog Rules
$e(\bar{X}) = e_1(\bar{X_2}) \vee e_2(\bar{X_2})$	$e(X_1, ..., X_n) :- e1(X_{1,1}, ..., X_{1,k}).$ $e(X_1, ..., X_n) :- e2(X_{2,1}, ..., X_{2,m}).$
$e(\bar{X}) = e_1(\bar{X_1})$ when $c[\bar{X_2}]$	$e(X_1, ..., X_n) = e_1(X_{1,1}, ..., X_{1,k}), c(X_{2,1}, ..., X_{2,m}).$
$e(\bar{X}) = e_1(\bar{X_1}) \wedge_{\leq t} e_2(\bar{X_2})$	$materialize(e'_1, keys(1,2), t).$ $materialize(e'_2, keys(1,2), t).$ $e'_1(X_{1,1}, ..., X_{1,k}) :- e_1(X_{1,1}, ..., X_{1,k}).$ $e'_2(X_{2,1}, ..., X_{2,m}) :- e_2(X_{2,1}, ..., X_{2,m}).$ $e(X_1, ..., X_n) = e_1(X_{1,1}, ..., X_{1,k}), e'_2(X_{2,1}, ..., X_{2,m}).$ $e(X_1, ..., X_n) = e_2(X_{2,1}, ..., X_{2,m}), e'_1(X_{1,1}, ..., X_{1,k}).$
$e(\bar{X}) = start(c[\bar{Y}])$	$e(X_1, ..., X_n) :- c\_ins(Y_1, ..., Y_m).$
$e(\bar{X}) = end(c[\bar{Y}])$	$e(X_1, ..., X_n) :- c\_del(Y_1, ..., Y_m).$
$c[\bar{X}] = c_1[\bar{X_1}] \wedge c_2[\bar{X_2}]$	$c(X_1, ..., X_n) :- c_1(X_{1,1}, ..., X_{1,k}), c_2(X_{2,1}, ..., X_{2,m}).$
$c[\bar{X}] = c_1[\bar{X_1}] \vee c_2[\bar{X_2}]$	$c(X_1, ..., X_n) :- c_1(X_{1,1}, ..., X_{1,k}).$ $c(X_1, ..., X_n) :- c_2(X_{2,1}, ..., X_{2,m}).$
$c[\bar{X}] = pred(v_1[\bar{Z_1}], ...v_p[\bar{Z_p}])$	$c(X_1, ..., X_n) :- v_1(Z_{1,1}, ..., Z_{1,m_1}, Val_1), ...,$ $v_p(Z_{p,1}, ..., Z_{p,m_p}, Val_p), pred(Val_1, ...Val_p).$
$c[\bar{X}] = [e_1(\bar{X_1}), e_2(\bar{X_2}))$	$c(X_1, ..., X_n) :- e_1(X_{1,1}, ..., X_{1,k}).$ $delete\ c(X_1, ..., X_n) :- e_2(X_{2,1}, ..., X_{2,m}), c(X_1, ..., X_2).$
$e(\bar{X}) \rightarrow \{v[\bar{Z}] := expr(v_1[\bar{Z_1}], ..., v_p[\bar{Z_p}])\}$	$v(Z_1, ..., Z_n, Val) :- v_1(Z_{1,1}, ..., Z_{1,m_p}, Val_1), ...,$ $v_p(Z_{p,1}, ..., Z_{p,m_p}, Val_p), Val := expr(Val_1, ...Val_p).$

Datalog Generation

Normalized MEDL rules are rewritten into location-agnostic Datalog rules

MEDL Rules	Corresponding Datalog Rules
$e(\bar{X}) = e_1(\bar{X}_2) \vee e_2(\bar{X}_2)$	$e(X_1, ..., X_n) : -e1(X_{1,1}, ..., X_{1,k}).$ $e(X_1, ..., X_n) : -e2(X_{2,1}, ..., X_{2,m}).$
$e(\bar{X}) = e_1(\bar{X}_1)$ when $c[\bar{X}_2]$	$e(X_1, ..., X_n) = e_1(X_{1,1}, ..., X_{1,k}), c(X_{2,1}, ..., X_{2,m}).$

Define condition $c(X_1,\ldots,X_n)$ over auxiliary variables $v_1[X_1]$ … $v_n[X_n]$

E.g. $c[x,y] = v_1[x] + v_2[y] > 5$ ⇒

c(X,Y) :- v1(X, Val1), v2(Y, Val2), Val1 + Val2 > 5.

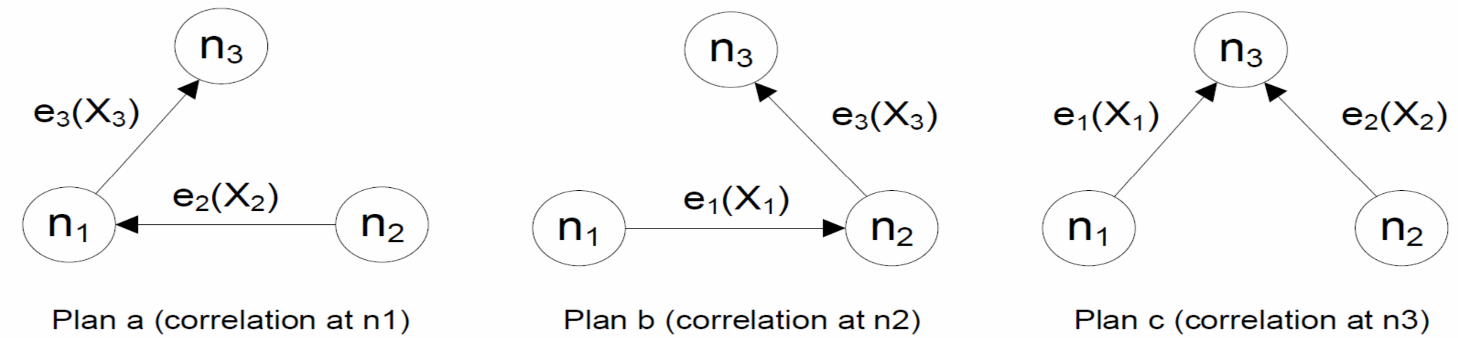
The rule is triggered by update events of any variable in the rule body

MEDL Rules	Corresponding Datalog Rules
$e(\bar{X}) = end(c[\bar{Y}])$	$e(X_1, ..., X_n) : -c\_del(Y_1, ..., Y_m).$
$c[\bar{X}] = c_1[\bar{X}_1] \wedge c_2[\bar{X}_2]$	$c(X_1, ..., X_n) : -c_1(X_{1,1}, ..., X_{1,k}), c_2(X_{2,1}, ..., X_{2,m}).$
$c[\bar{X}] = c_1[\bar{X}_1] \vee c_2[\bar{X}_2]$	$c(X_1, ..., X_n) : -c_1(X_{1,1}, ..., X_{1,k}).$ $c(X_1, ..., X_n) : -c_2(X_{2,1}, ..., X_{2,m}).$
$c[\bar{X}] = pred(v_1[\bar{Z}_1], ...v_p[\bar{Z}_p])$	$c(X_1, ..., X_n) : -v_1(Z_{1,1}, ..., Z_{1,m_1}, Val_1), ...,$ $v_p(Z_{p,1}, ..., Z_{p,m_p}, Val_p), pred(Val_1, ...Val_p).$
$c[\bar{X}] = [e_1(\bar{X}_1), e_2(\bar{X}_2))$	$c(X_1, ..., X_n) : -e_1(X_{1,1}, ..., X_{1,k}).$ $delete\ c(X_1, ..., X_n) : -e_2(X_{2,1}, ..., X_{2,m}), c(X_1, ..., X_2).$
$e(\bar{X}) \rightarrow \{v[\bar{Z}] := expr(v_1[\bar{Z}_1], ..., v_p[\bar{Z}_p])\}$	$v(Z_1, ..., Z_n, Val) : -v_1(Z_{1,1}, ..., Z_{1,m_p}, Val_1), ...,$ $v_p(Z_{p,1}, ..., Z_{p,m_p}, Val_p), Val := expr(Val_1, ...Val_p).$

Optimized NDlog Generation

- **Execution location for rules with inputs from multiple nodes?**

Options for executing $e_3(X_3) = e_1(X_1) \wedge_{<=t} e_2(X_2)$



- **Cost-based query optimization**

Plan a: $|e_2| * s_{e_2} + r_{e_1,e_2} * |e_1| * |e_2| * s_{e_3}$

Plan b: $|e_1| * s_{e_1} + r_{e_1,e_2} * |e_1| * |e_2| * s_{e_3}$

Plan c: $|e_1| * s_{e_1} + |e_2| * s_{e_2}$

- **Existing optimization techniques in Database literatures**
  - Dynamic programming, heuristics, *adaptive query optimization*

Outline of Talk

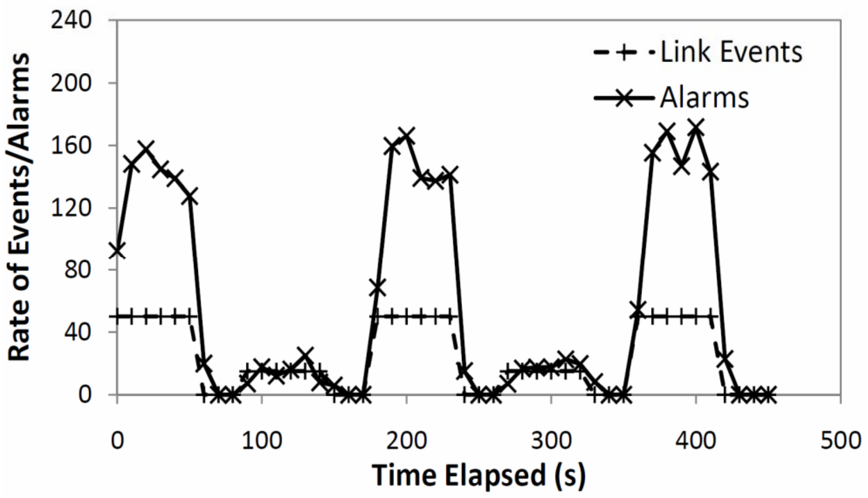
- Introduction
- Background and Motivation
- DMaC Architecural Overview
- Compilation to Declarative Networking Queries
- Experimental Evaluation
- Conclusion & Future Work

Experimental Setup

- **Based on P2 declarative networking system**
- **Workload**
  - Path-vector – shortest paths between all pairs of nodes
  - Monitor route persistence property. When violated, raise alarm
  - Emulate changes to the network topology
    - 60 seconds high churn (50 link updates per second)
    - 60 seconds low churn (15 link updates per second)
- **Testbed**
  - A local cluster with 15 quad-core machines
  - Total 120 p2 instances (eight per machine)

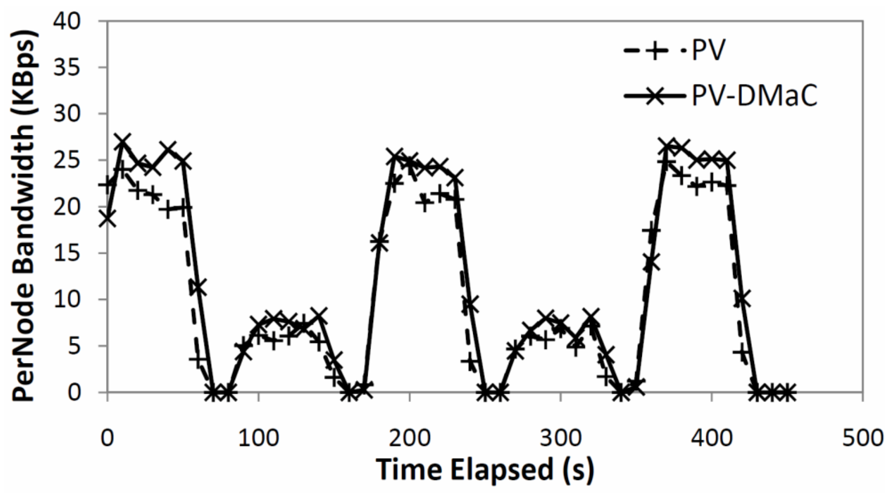
Feasibility Study of DMaC

Correlation between events and alarms



- Experimentally validate the correctness of the DMaC implementation.
- In high churn, persistence property is more likely to be violated
- In low churn, the number of alarms drops significantly

Bandwidth Utilization



- Study the additional overhead incurred by monitoring rules
- Incur an 11% increase for *PV-DMaC* in bandwidth utilization
- Absolute increase is 2.5KBps, well-within capacity of typical network connections

Conclusion & Future Work

- **Unifies two body of work: MaC + NDlog**
  - A framework of distributed runtime verification and its deployment
  - Compilation of formal specifications to distributed queries
  - Proof-of-concept experimental evaluation to validate feasibility
- **Future Work**
  - Cost-based optimization for distribution and execution plan
    - Accurate cost estimation
    - Efficient search for optimal plan
    - Adaptive optimization according to changes of settings

Thank You …